Escuela Politécnica Superior

Algoritmos Paralelos para la corrección de ruido mixto gaussiano-impulsivo en imágenes digitales

Máster Universitario en Ingeniería Informática

Trabajo Fin de Máster

Autor:
Juan Bautista Pérez Mingot

Tutor:
Josep Arnal García

Universitat d'Alacant
Universidad de Alicante

Enero 2017

Dedicado a Vicenta Mingot, mi madre,

quien me enseñó a sumar antes de que supiese leer.

In Memoriam

Índice de contenido

1. Introducción

1.1. Motivación

Las imágenes digitales, durante su proceso de adquisición, transmisión y almacenamiento, pueden corromperse mediante ruido. Así pues, un problema importante del procesamiento de imágenes es eliminar dicho ruido preservando algunas características como las texturas, bordes y detalles. Dos de los problemas más comunes son el ruido gaussiano, introducido durante el proceso de adquisición de la imagen [1], y el ruido impulsivo, introducido en la imagen por errores de transmisión o fallos de almacenamiento [2].

Se han revisado diferentes métodos para eliminar tanto el ruido gaussiano [1]–[7], como el ruido impulsivo [8]–[18].

En este trabajo se plantea una solución mixta basada en lógica fuzzy. Se afronta la corrección de imágenes en color corruptas con ruido mixto gaussiano e impulsivo.

El algoritmo paralelo introducido en este trabajo se basa en el filtro de dos etapas basado en lógica fuzzy para imágenes con ruido mixto gaussiano-impulsivo introducido en [19] y en el filtro basado en lógica fuzzy para imágenes con ruido mixto gaussiano-impulsivo introducido en [20].

Los resultados experimentales ponen de manifiesto que estas técnicas de filtrado muestran una gran competitividad respecto a otros métodos del estado del arte actual.

Por otro parte, los cálculos implícitos en estos algoritmos presentan un alto grado de paralelismo. Por estos motivos, en este trabajo se presenta un algoritmo paralelo basado en estos métodos, con el objetivo de aprovechar sus buenos resultados en calidad de imagen para la detección y corrección de ruido mixto gaussiano-impulsivo, mejorando su rendimiento y obteniendo tiempos de ejecución mejorados en relación con su ejecución secuencial.

El algoritmo paralelo presentado en este trabajo, basado en lógica fuzzy para la detección y corrección de ruido mixto gaussiano-impulsivo, va a ser presentado por su autor en el Congress on Numerical Methods in Engineering - CMN 2017, organizado por la Sociedad Española de Métodos Numéricos en Ingeniería y la Associação Portuguesa de Mecânica Teórica, Aplicada e Computacional, a celebrar en Valencia (Spain) del 3 al 5 de Julio de 2017 [21].

1.2. Método simple basado en lógica fuzzy para la corrección de ruido mixto gaussiano-impulsivo en imágenes digitales a color

Conforme se ha visto anteriormente, existen diversas soluciones de filtrado para reducir los diferentes tipos de ruido por separado, y además, la mayoría de ellos se aplican para imágenes de escala de grises.

La manera más simple de reducir el ruido mixto en una imagen digital es aplicar consecutivamente varios métodos específicos (usualmente dos) para cada tipo de ruido en la imagen. Sin embargo, la aplicación de dichos filtros podría reducir drásticamente la eficiencia computacional de los procesos, lo que implica que esta solución no puede ser práctica para aplicaciones reales. Por lo tanto, es más interesante diseñar filtros para eliminar el ruido mixto gaussiano-impulsivo en un mismo algoritmo.

La teoría difusa es útil para construir soluciones simples, eficientes y efectivas para este problema. En este trabajo, se utiliza un método difuso para reducir el ruido mixto gaussiano-impulsivo en imágenes en color. Dicho método utiliza una única operación de filtrado, *la media de pesos ponderada*, utilizándose un sistema de reglas difusas para asignar los pesos en cada píxel, de modo que ambos tipos de ruido se reduzcan y la estructura de la imagen sea preservada [20].

Los resultados obtenidos por los autores del filtro, demuestran que su método es muy competitivo con los actuales filtros de vanguardia, respecto de la calidad de las imágenes, pero el tiempo de cómputo que obtienen es muy elevado, por lo que ha sido el motivo por el que se ha utilizado dicho filtro para la realización de este trabajo.

1.3. Proceso de resolución

Se ha probado este algoritmo desarrollando programas paralelos, obteniéndose unos buenos resultados de *speedup* (en algunos casos, ya que es un método iterativo) con respecto al número de procesadores utilizados.

Para establecer una medida cuantitativa de la mejora obtenida respecto a las imágenes corruptas por el ruido, se ha utilizado el PSNR (Peak Signal to Noise Ratio) [22]–[23].

Este tipo de estadística nos da una medida de la calidad de la reconstrucción, es decir, de la similitud entre la imagen original y la obtenida después de aplicarle el filtro.

1.4. Organización del documento

Este documento está organizado por capítulos, con el siguiente detalle:

Capítulo 1. Introducción: exposición de motivos del presente trabajo.

Capítulo 2. Organización secuencial del proyecto: explicación de la secuencia de actividades realizadas.

Capítulo 3. Algoritmos paralelos para la corrección de ruido mixto gaussiano-impulsivo en imágenes digitales: introduce y explica el algoritmo paralelo basado en el método “A Simple Fuzzy Method to Remove Mixed Gaussian-Impulsive Noise From Color Images” [20].

Capítulo 4. Experimentos planteados: explicación de los experimentos y ejecuciones realizadas.

Capítulo 5. Resultados: exposición de resultados obtenidos.

Capítulo 6. Conclusiones: muestra las conclusiones a las que se ha llegado.

Capítulo 7. Líneas de trabajo futuras: expone futuras ampliaciones al trabajo mediante otras técnicas de paralelización, y aplicación del mismo a diferentes sectores de investigación.

Capítulo 8. Referencias: lista las bibliografías y trabajos citados en este documento.

2. Organización secuencial del proyecto

Para una correcta realización del presente trabajo se ha dividido en procesos o estadios el conjunto de actividades a realizar en el proyecto.

En un primer estadio, se hizo un estudio del algoritmo y se implementó de manera secuencial utilizando Matlab, comprobándose que los resultados fueran óptimos y acordes con los obtenidos por los autores de dicho método.

Seguidamente, se adaptó el mismo al lenguaje Fortran, adaptando y modificando lo necesario para el recorrido de las matrices “por columnas” para un rendimiento más efectivo [24], y comprobando asimismo que los resultados fueran óptimos.

Después de efectuar un estudio del estado del arte acerca de la corrección de ruidos, la comparación de imágenes y las medidas a utilizar para la misma, se buscó un conjunto de imágenes para las pruebas a efectuar.

Tras manipular las imágenes, mediante un algoritmo en Matlab, introduciéndoles tanto ruido impulsivo como gaussiano, con diferentes combinaciones de valores, se extrajo la información de dichas imágenes descomponiéndolas en componentes RGB, también en Matlab, para hacer uso de la misma en la ejecución de los tests.

A continuación, se realizó la implementación del algoritmo para paralelizarlo, utilizando el conjunto de librerías OpenMP [25] sobre el algoritmo hecho en

Fortran. OpenMP es una librería muy potente que aplica paralelización a la ejecución de tareas en entornos de memoria compartida.

Finalmente, dentro del desarrollo y prueba, se diseñó una estructura para la ejecución de las pruebas y para recuperar los datos obtenidos para su correcto análisis.

Realizadas las pruebas y recogidos los datos, se procedió a efectuar un análisis estadístico para la extracción de conclusiones finales.

Para finalizar el trabajo, se redactó la memoria del Trabajo Fin de Máster.

Toda la organización secuencial del proyecto se puede observar en la Figura 1.

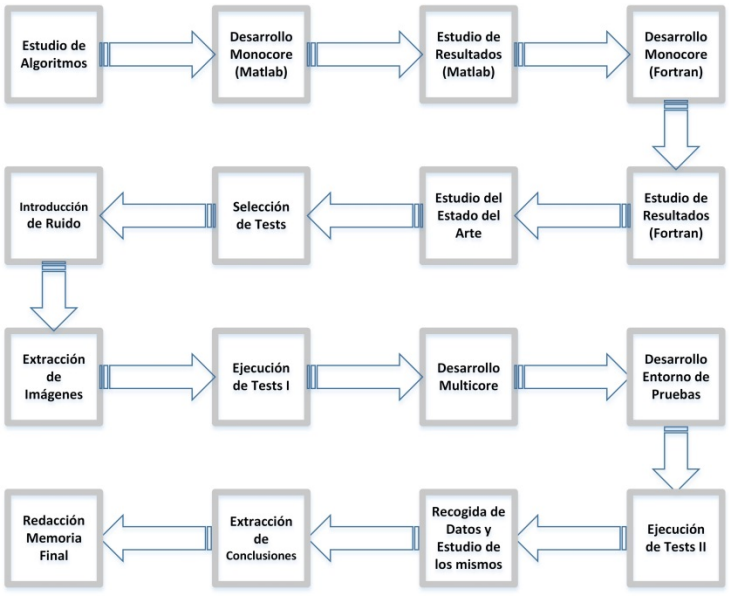


Figura 1. Grafo secuencial de actividades del proyecto. (elaboración propia)

3. Algoritmos paralelos para la corrección de ruido mixto gaussiano-impulsivo en imágenes digitales

3.1. Introducción

Para la correcta paralelización de los algoritmos, en primer lugar vamos a considerar la descomposición del dominio Ω de la imagen en P subdominios $\{\Omega_i\}_{i=1..P}$, siendo P el número de unidades de procesamiento utilizadas, a fin de describir correctamente los algoritmos paralelos, así como la forma en que los píxeles son asignados a cada elemento de cómputo. La Figura 2 muestra un ejemplo de la descomposición del dominio utilizada en los experimentos.

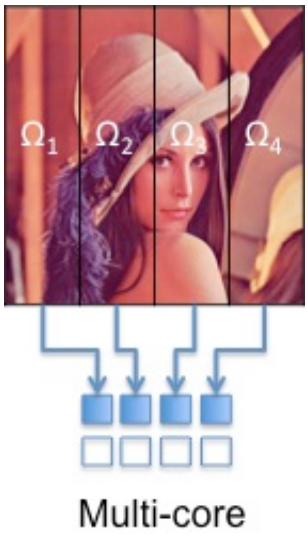


Figura 2. Descomposición en subdominios de la imagen. Ejemplo de imagen distribuida en 4 cores. (elaboración propia)

Es evidente que en la versión secuencial del algoritmo P=1, obteniéndose de la descomposición un único subdominio, $\Omega=\Omega_1$.

Se ha establecido que a los algoritmos se les aplicará el paralelismo siguiendo sus propias características intrínsecas, por lo que en el entorno paralelo tendremos un subdominio de Ω por cada unidad de procesamiento disponible, quedando, por tanto, la imagen subdividida en un número de subdominios igual al número de unidades de procesamiento. En el ejemplo de la Figura 2, tenemos que P=4 y $\Omega=\Omega_1\cup\Omega_2\cup\Omega_3\cup\Omega_4$.

Debido a las características propias del lenguaje Fortran, utilizado en el desarrollo del trabajo, ya que este lenguaje organiza los elementos de una matriz almacenándolos en direcciones consecutivas de memoria por columnas [26], y por tanto, obteniendo mejores rendimientos en los recorridos verticales de dichas matrices, se ha optado por esta manera de descomponer la imagen en subdominios.

Si se hubiese optado por una descomposición de la imagen *"por filas"*, se habrían obtenido peores tiempos de ejecución [24].

3.2. Algoritmo paralelo para la corrección de ruido mixto gaussiano-impulsivo en imágenes de color

El algoritmo utilizado es un método simple para eliminar el ruido mixto gaussiano-impulsivo de imágenes en color basado en la lógica difusa.

El funcionamiento del mismo se basa en que cada píxel de la imagen con ruido se filtra sólo una vez usando la misma operación: un promedio ponderado simple en relación a sus píxeles vecinos en una ventana de filtrado.

La naturaleza adaptativa del método se basa en la forma en que se calculan los pesos que están involucrados, para lo cual utiliza un sistema basado en reglas fuzzy.

Este sistema difuso toma como entrada dos fuentes de información sobre los píxeles en la ventana de filtrado:

1) el grado de ruido (desde el punto de vista impulsivo) que se calcula usando un método estadístico, y
2) los grados de similitud entre el píxel central y el resto de los píxeles vecinos en la ventana de filtrado.

A partir de esta información, el método propuesto calcula los pesos que nos permiten procesar cada píxel de una manera apropiada, reduciendo el ruido y preservando las estructuras de la imagen apropiadamente.

Hay que recalcar que el algoritmo es iterativo, en cada iteración calcula el nuevo PSNR, y si se ha mejorado el mismo, vuelve a iterar hasta que el nuevo PSNR no es mejor que el anterior.

3.3. Descripción del algoritmo paralelo para la corrección de ruido mixto gaussiano-impulsivo en imágenes de color

3.3.1. Introducción y objetivo

Sea $\mathcal{F}$ la imagen en color a procesar, y sea W una ventana de filtrado, de tamaño $n \times n$ ($n = 3, 5, ...$), centrada en el píxel $\mathbf{F}_0$ bajo procesamiento. En este trabajo se ha optado por una ventana 3x3. Los vectores en $\mathcal{F}$ se denotan como $\mathbf{F}_i = (F_i^R, F_i^G, F_i^B)$, como es habitual en el espacio de colores RGB.

El método propuesto, denominado *simple fuzzy rule filter* (SFRF), consiste en reemplazar un píxel $\mathbf{F}_0$ de la imagen por un píxel $\widehat{\boldsymbol{F}}_0$, el cual es un promedio ponderado de ciertos píxeles seleccionados en W, denotados por $F_0, F_1, ..., F_m$, y viene dado por la siguiente formula

$$\widehat{\boldsymbol{F}}_0 = \left(\sum_{i=0}^{m} w_i * \boldsymbol{F}^i\right) / \left(\sum_{i=0}^{m} w_i\right)$$

donde los pesos $w_i \in [0, 1]$ se obtienen mediante defuzzificación utilizando inferencia de lógica difusa por un sistema difuso.

La adaptabilidad del método se da por el uso de estos pesos y la forma en que se calculan. Este conjunto de pesos es diferente para cada ventana de filtrado y depende de las características locales observadas, lo que permite procesar

adecuadamente ambos tipos de píxeles ruidosos (gaussiano e impulsivo), así como las estructuras de la imagen originales.

En las siguientes secciones de este capítulo, se detalla cómo obtener estos pesos para cada ventana de filtrado.

Se usa principalmente dos fuentes de información: primero, el ruido, desde el punto de vista impulsivo, de cada píxel en la imagen; y segundo, las similitudes que se observan entre el píxel bajo procesamiento y el resto de los píxeles en la ventana. A partir de esta información, un sistema basado en reglas difusas obtiene los pesos a través de inferencia difusa.

3.3.2. Cálculo del ruido en los píxeles de la imagen

En el primer paso, evaluamos cuán ruidoso es cada píxel de imagen. Por lo tanto, asignamos un grado de certeza $\delta(\mathbf{F}_i)$ para la declaración vaga "$\boldsymbol{F}_i$ *es ruidoso*", a cada F_i de la siguiente forma.

Ordenamos los píxeles F_j en una ventana W' centrada en F_i, que también se toma, por simplicidad, de tamaño n×n en la forma $F_{(0)}, F_{(1)}, \ldots, F_{(n2\ -1)}$ de acuerdo con una medida de distancia ϱ, tal que $\varrho(F_i, F_{(0)}) \leq \varrho(F_i, F_{(1)}) \leq \cdots \leq \varrho(F_i, F_{(n2\ -1)})$, donde obviamente $F_{(0)}=F_i$.

Como medida de distancia ϱ, utilizamos la Norma infinito L_∞, dada por

$$L_\infty(F_i, F_j) = max\{|F_i^R - F_j^R|, |F_i^G - F_j^G|, |F_i^B - F_j^B|\}$$

por su alta sensibilidad en la detección de ruido impulsivo.

A continuación, consideramos los s+1 primeros píxeles $F_{(0)}$, $F_{(1)}$, …, $F_{(s)}$ y computamos el ROD_s (medida para calcular la cercanía de un píxel a otro en cuanto a parecido, no a distancia) para el píxel F_i,

$$ROD_s = \sum_{j=0}^{s} L_\infty\left(F_i, F_{(j)}\right)$$

Dado que $F_i=F_{(0)}$, entonces $L_\infty\left(F_i, F_{(0)}\right) = 0$, y ROD_s tomará valores en el intervalo [0, 255].

Un valor bajo de $ROD_s(F_i)$ significa que los s+1 píxeles $F_{(j)}$ seleccionados en W' son cercanos a F_i, lo que a su vez indica que se espera que F_i sea libre de ruido. Asimismo, los valores más altos de $ROD_s(F_i)$ indican un mayor grado de ruido para F_i, ya que no se encuentran píxeles cercanos a él. Los autores del método ajustan el parámetro s=2.

Declaramos $x=ROD_s(F_i)$ y definimos el grado de certidumbre $\delta(F_i)$ para la afirmación vaga "*F_i es ruidoso*", dado por

$$\delta(F_i) = f(x) = \begin{cases} 0, & x \leq k_1 \\ \dfrac{x - k_1}{k_2 - k_1}, & k_1 < x < k_2 \\ 1, & k_2 \leq x \end{cases}$$

donde los autores del método ajustan los valores $k_1=0,5*ROD_{max}$ y $k_2=0,6*ROD_{max}$, siendo $ROD_{max}=max\{ROD_s(F_i) : F_i \in \digamma\}$

Finalmente, asignamos a cada píxel F_i de $\digamma$ un grado de certidumbre para la afirmación vaga "*F_i no es ruidoso*". Representando la negación por el operador involutivo difuso, será dado por 1 - $\delta(F_i)$. Los correspondientes conjuntos fuzzy f y 1-f, los cuales se definen en [0, 255], pueden verse en la Figura 3(a).

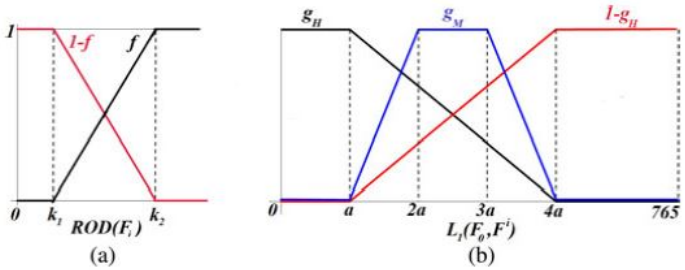


Figura 3. (a) Grado de ruidosidad de un pixel F_i como una función f de $ROD_s(F_i)$. (b) Similitud de F^i con respecto a F_0 como función de $L_1(F_0, F^i)$. [20]

3.3.3. Similitud entre el píxel bajo proceso y el resto de píxeles en la ventana de filtrado

En el segundo paso del algoritmo, estamos interesados en analizar la similitud entre el píxel bajo proceso F_0 y el resto de los píxeles en la ventana de filtrado W.

Para medir la similitud entre dos píxeles, ahora utilizamos la Norma uno L_1, dada por

$$L_1(F_i, F_j) = \left|F_i^R - F_j^R\right| + \left|F_i^G - F_j^G\right| + \left|F_i^B - F_j^B\right|$$

En este paso, se prefiere utilizar la métrica L_1 en vez de L_∞, porque esta última no es adecuada para medir la similitud, dado que se centra más en las mayores diferencias que se encuentran entre los componentes de color y no en su similitud. La Norma L_1 es mejor para medir la similitud porque en el cálculo toma en consideración los tres compontes RGB de los dos píxeles en comparación y las diferencias entre los componentes no son enfatizadas, a diferencia de otras métricas como L_∞ y la Norma euclídea (L_2).

Ahora, usando las similitudes observadas, asignaremos a algunos píxeles seleccionados de W, denotados por F_i, un grado de certeza en las declaraciones vagas siguientes: *la similitud entre F_i y F_0 es "alta", "media" y "baja"*, denotadas por $\mu_H(F_0, F^i)$, $\mu_M(F_0, F^i)$ y $\mu_L(F_0, F^i)$, respectivamente.

Sin embargo, previamente, seleccionamos los píxeles que estarán involucrados en la operación de filtrado. Usando la distancia $L_1(F_0, F_j)$ entre cada píxel F_j de W y el píxel bajo procesamiento F_0, introducimos un nuevo ordenamiento para los n^2 píxeles de W en el conjunto ordenado $W^*=\{F^0, F^1, ..., F^{n^2-1}\}$ tal que $L_1(F_0,F^0) \leq L_1(F_0,F^1) \leq \cdots \leq L_1(F_0,F^{n^2-1})$, donde, obviamente, $F_0=F^0$. Luego, seleccionamos los primeros m+1 píxeles $F^0, ..., F^m$ para evitar

que los píxeles muy diferentes de F_0 participen en el filtrado. Los autores del método ajustan el parámetro m=7.

Ahora, para asignar los grados de certidumbre de las tres proposiciones vagas anteriores, definimos las funciones siguientes:

Declaramos $x=L_1(F_0, F^i)$, y definimos $\mu_H(F_0, F^i)=g_H(x)$ por

$$g_H(x) = \begin{cases} 1, & x \leq a \\ -\dfrac{x}{(3a)+\frac{4}{3}}, & a < x < 4a \\ 0, & 4a < x \end{cases}$$

Usando la negación fuzzy, asignamos $\mu_L(F_0, F^i)= 1 - \mu_H(F_0, F^i)$.

La certidumbre para $\mu_M(F_0, F^i)=g_M(x)$ es

$$g_M(x) = \begin{cases} \frac{x-a}{a}, & a<x<2a \\ 1, & 2a\leq x\leq 3a \\ \frac{4a-x}{a}, & 3a<x<4a \\ 0, & en\ otro\ caso \end{cases}$$

Los autores del método relacionan el parámetro a con σ, ajustando dicho parámetro $a = 0{,}998\sigma + 1{,}960$. Los correspondientes conjuntos fuzzy g_H , 1-g_H, y g_M, definidos en $[0, 3\cdot 255]$, pueden observarse en la Figura 3(b).

3.3.4. Sistema difuso y computación de pesos

Para calcular los pesos que están involucrados en la operación de filtrado, ahora usamos un sistema basado en reglas difusas y una inferencia difusa.

El sistema difuso utiliza las declaraciones vagas que se han descrito en las secciones anteriores para decidir si cada peso en la fórmula de la sección 3.3.1. debe ser grande, medio o pequeño. Finalmente, se usa la defuzzificación para obtener el valor particular para cada peso.

El objetivo de las reglas en el sistema difuso se puede resumir en dos ideas principales:

1) los píxeles que son ruidosos se deben asignar a un peso pequeño, y
2) los píxeles que están libres de ruido sólo pueden asociarse con un peso mayor si son similares al píxel central o si el píxel central es ruidoso.

Esta última idea, para los diferentes casos que se encuentran, se resume en el siguiente árbol de reglas difusas, que son las utilizadas en el algoritmo para defuzzificar y obtener los valores de cada peso:

1) IF (F^i es no ruidoso AND
   F_0 es ruidoso AND
   la similitud entre F^i y F_0 es media)
   THEN w_i es un peso medio
2) IF (F^i es no ruidoso AND
   F_0 es ruidoso AND

la similitud entre F^i y F_0 es baja) OR

(F^i es no ruidoso AND

F_0 es no ruidoso AND

la similitud entre F^i y F_0 es alta)

THEN w_i es un peso grande

3) IF (F^i es ruidoso) OR

(F^i es no ruidoso AND

F_0 es ruidoso AND

la similitud entre F^i y F_0 es alta) OR

(F^i es no ruidoso AND

F_0 es no ruidoso AND

la similitud entre F^i y F_0 es media) OR

(F^i es no ruidoso AND

F_0 es no ruidoso AND

la similitud entre F^i y F_0 es baja)

THEN w_i es un peso pequeño

Antes de realizar el proceso de inferencia fuzzy, necesitamos definir los conjuntos difusos correspondientes a los consecuentes de las reglas difusas.

Cada peso $w_i \in [0,1]$ está asociado con un grado de certidumbre en las declaraciones vagas "*w_i es un peso grande*", "*w_i es un peso medio*" y "*w_i es un peso pequeño*", que se denotan por $\nu_L(w_i)$, $\nu_M(w_i)$ y $\nu_S(w_i)$, respectivamente.

Los conjuntos difusos ν_L, ν_M, y ν_S se representan en la Figura 4, donde se han seleccionado las funciones de pertenencia difusa de forma triangular para simplificar el proceso de defuzzificación.

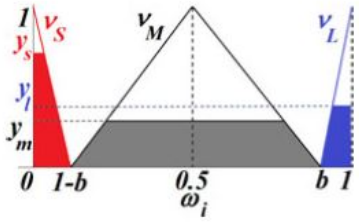


Figura 4. Conjuntos fuzzy ν_L, ν_M y ν_S. [20]

Ahora, para calcular el grado de certidumbre de los antecedentes de las reglas difusas, siguiendo el procedimiento usual en la lógica difusa, aplicamos la operación de conjunción AND y la operación de disyunción OR por medio de una t-norma * y su s-norma asociada *', respectivamente. En este trabajo, se usa el producto usual como la t-norma y la adición probabilística como la s-norma, S(A,B) = A+B – A*B.

Las certezas de los antecedentes se asignan a los consecuentes, y finalmente, por defuzzificación, se obtiene el peso w_i del pixel F^i.

Al hacerlo, se ha utilizado la técnica centroide, o centro de gravedad (COG), que es el método más popular de defuzzification [27]–[29], del modo siguiente:

Supongamos que para un píxel F^i, y_m, y_l y y_s son los grados de certeza de los consecuentes en las reglas 1, 2 y 3 anteriores, respectivamente. En la Figura 3, consideremos los triángulos definidos por ν_M, ν_L y ν_S y las funciones constantes $y=y_m$, $y=y_l$, $y=y_s$. Tomando la superficie en cada uno de los triángulos mencionados y bajo cada una de las tres funciones constantes, respectivamente, se construyen tres trapecios. La línea poligonal que está constituida por las cimas y los lados de estos trapecios determina un conjunto difuso A en [0,1] que es integrable en el sentido de Riemann (clásico). La abscisa del COG del área bajo A es el peso w_i.

Por lo tanto, $w_i = \left(\int_0^1 x \cdot A(x)\, dx\right) / \left(\int_0^1 A(x)\, dx\right)$.

Los pesos para los píxeles $\mathbf{F}^i$ para i = 0, 1, ..., m nos permiten aplicar la fórmula de la sección 3.3.1. para obtener el deseado pixel sin ruido $\widehat{\boldsymbol{F}}_0$.

4. Experimentos planteados

El rendimiento del presente algoritmo se probó en dos entornos multicore de distinta configuración.

- Multicore 1: Intel Xeon E5320 (8 cores), 1.86 GHz, 8 GB RAM, Linux Ubuntu 8.04.1, GNU Fortran.
- Multicore 2: 2 x Intel Xeon X5660 (12 cores), 2.8 GHz, 48 GB RAM, Linux Centos 5.6, Intel Fortran.

Las imágenes utilizadas en las pruebas son de dominio público y su tamaño no ha sido modificado, pudiendo verse las mismas, con su resolución y referencia en la Tabla 1.

Para la correcta aplicación del algoritmo cada imagen se descompuso en sus componentes RGB mediante un *script* realizado en Matlab, para seguidamente proceder a la inducción de ruido gaussiano e impulsivo.

Se ha optado por el ruido gaussiano clásico blanco y por ruido impulsivo de valor aleatorio, introduciéndose en la imagen mediante otro *script* Matlab.

Para cada imagen se diseñaron tres configuraciones de ruido gaussiano (σ) e impulsivo (ρ) diferente, con los siguientes valores de ruido: A) σ=10 ρ=0,1, B) σ=20 ρ=0,2 y C) σ=30 ρ=0,3.

Se guardaron las imágenes generadas con ruido, y se descompuso cada una en sus correspondientes componentes RGB mediante el mismo *script* Matlab descrito anteriormente.

El motivo de su descomposición en componentes RGB es por el diseño del algoritmo en Fortran y su utilización de matrices de datos.

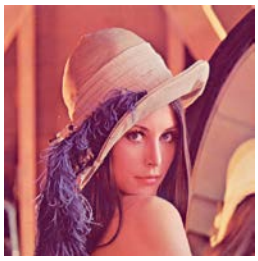
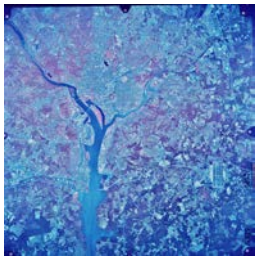
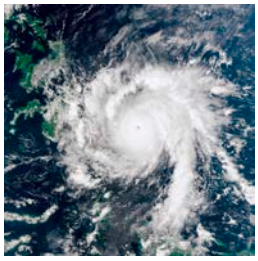
		
Lena [32] 512x512	Busto [31] 1024x1024	Caravana [33] 1080x1080
		
Wash-IR [35] 2250x2250	Zapatos [34] 3000x3000	Typhoon Bopha [36] 5000x5000

Tabla 1. Imágenes utilizadas en las pruebas. (elaboración propia)

Las pruebas se ejecutaron tanto en secuencial como en paralelo y teniendo en cuenta el número de cores disponibles en cada entorno.

Como resultado de cada prueba, el algoritmo devuelve tres archivos con los componentes RGB de la imagen restaurada.

Del total de las pruebas se han obtenido más de 90 GB de datos; solo por poner un ejemplo, en la restauración de la imagen Typhoon Bopha, imagen de 5000x5000 píxeles, se obtuvieron tres archivos con la descomposición RGB de 650 MB cada uno por prueba, y así para cada una de las tres configuraciones de ruido aplicadas y para cada ejecución con diferente número de procesadores y diferente máquina, en total, solo para esta imagen se obtuvieron unos 25 GB de datos.

Con los tres archivos obtenidos se ha aplicado un *script* Matlab, dando como resultado la imagen final reconstruida.

Asimismo, en cada una de las ejecuciones se midió el tiempo de ejecución, tanto en secuencial como en paralelo con diferentes configuraciones, 1 core, 2 cores, y así sucesivamente hasta el total de cores de la máquina.

Para comparar la calidad de las imágenes restauradas se ha aplicado el algoritmo PSNR (Peak Signal to Noise Ratio), que nos da una medida de la calidad de la reconstrucción, es decir, de la similitud entre la imagen original y la obtenida después de aplicarle el filtro.

Para cuantificar el rendimiento del algoritmo se ha considerado el *speedup* paralelo S_p definido como

$$S_p = \frac{T_{seq}}{T_p}$$

siendo T_{seq} el tiempo de ejecución del algoritmo secuencial y T_P el tiempo de ejecución del algoritmo paralelo utilizando P unidades de procesamiento.

Dado el alto volumen de experimentos realizados, en la presente memoria se exponen tan solo una parte representativa de ellos.

5. Resultados

Los resultados obtenidos en relación al PSNR han sido muy buenos, obteniéndose en muchos casos una mejora en la calidad de la imagen reconstruida, con respecto a la imagen con ruido, de hasta el 160%.

Examinando de forma visual las imágenes reconstruidas, se puede apreciar a simple vista, que la mejora al aplicar el algoritmo es muy buena.

En la Figura 5 se expone el resultado de la aplicación del algoritmo sobre la imagen Lena, con unos valores de ruido gaussiano σ=10 e impulsivo ρ=0,1, dando un PSNR de 28,54 en 16 iteraciones y observándose que la reconstrucción es casi perfecta.

Figura 5. Detalle de imagen Lena, original (izquierda), con ruido gaussiano σ=10 e impulsivo ρ=0,1 (centro) y restaurada (derecha). PSNR 28,54. 16 iteraciones. (elaboración propia)

En la Figura 6 se expone el resultado de la aplicación del algoritmo sobre la imagen Busto, con unos valores de ruido gaussiano σ=20 e impulsivo ρ=0,2,

dando un PSNR de 28,46 en 24 iteraciones y observándose que la reconstrucción es casi perfecta pese al valor más alto de ruido.

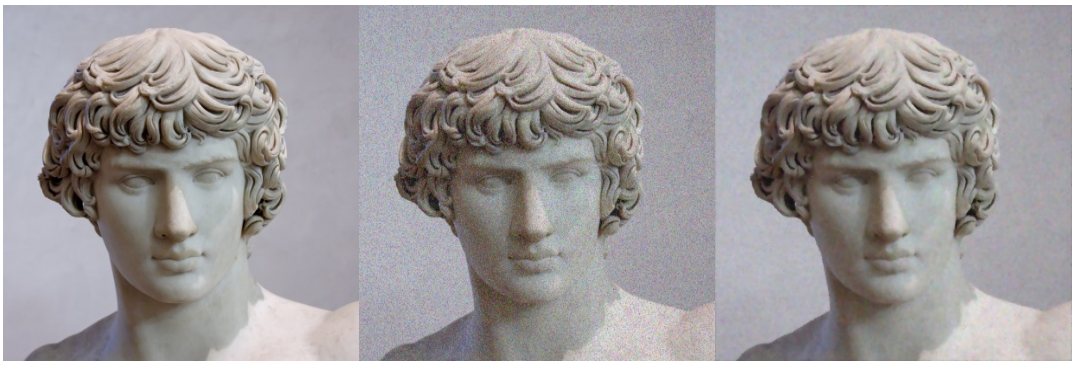


Figura 6. Imagen Busto, original (izquierda), con ruido gaussiano σ=20 e impulsivo ρ=0,2 (centro) y restaurada (derecha). PSNR 28,46. 24 iteraciones. (elaboración propia)

En la Figura 7 se expone el resultado de la aplicación del algoritmo sobre la imagen Zapatos, con unos valores de ruido gaussiano σ=30 e impulsivo ρ=0,3, dando un PSNR de 19,05 en 26 iteraciones y observándose el detalle de la etiqueta con el código de barras que ha sido reconstruido con alta calidad pese a los valores altísimos de ruido.

Figura 7. Detalle de Imagen Zapatos, original (izquierda), con ruido gaussiano σ=30 e impulsivo ρ=0,3 (centro) y restaurada (derecha). PSNR 19,05. 26 iteraciones. (elaboración propia)

A continuación se muestran dos gráficas con la evolución e incremento del PSNR. En la Figura 8 al aplicar el algoritmo sobre la Imagen Typhoon Bopha, con unos valores de ruido gaussiano σ=10 e impulsivo ρ=0,1, se observa que en 22 iteraciones pasa de un PSNR 19,82 a un PSNR 29,73, un 150% de incremento, y en la Figura 9 al aplicar el algoritmo sobre la Imagen Busto, con unos valores de ruido gaussiano σ=30 e impulsivo ρ=0,3, se observa que en 23 iteraciones pasa de un PSNR 15,60 a un PSNR 25,10, un 160% de incremento.

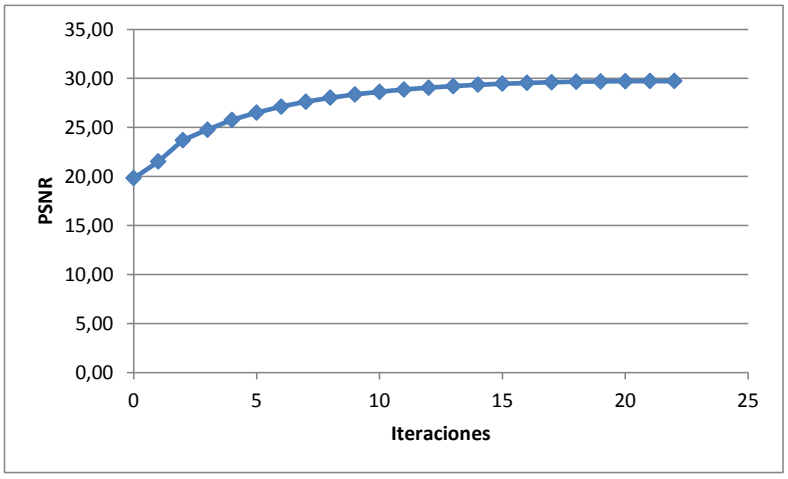


Figura 8. Evolución del PSNR en la restauración de la Imagen Typhoon Bopha en 22 iteraciones. (elaboración propia)

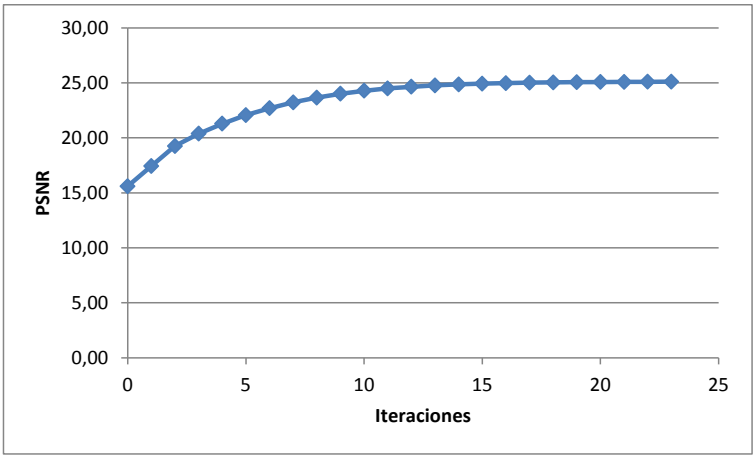


Figura 9. Evolución del PSNR en la restauración de la Imagen Busto en 23 iteraciones. (elaboración propia)

En la Figura 10 se presenta el *speedup* obtenido, tanto en el Multicore 1 como en el Multicore 2, para la imagen Lena, con unos valores de ruido gaussiano σ=10 e impulsivo ρ=0,1. Se puede observar que en el Multicore 2 se obtiene

una alta eficiencia. Por el contrario, en el Multicore 1, más antiguo y menos potente, el tiempo de cómputo es mayor, traduciéndose en una menor eficiencia.

En las pruebas se obtuvo, a mismo número de procesadores, ocho para cada entorno, un tiempo de 10,27 segundos en el Multicore 1 y 2,47 segundos en el Multicore 2, para el computo de las 16 iteraciones que realiza el algoritmo.

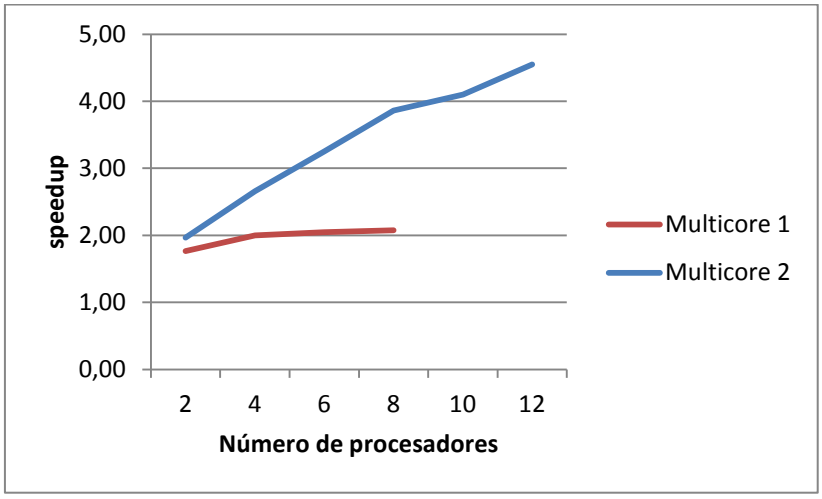


Figura 10. *speedup* obtenido al incrementar las unidades de procesamiento. Imagen Lena. (elaboración propia)

En cuanto a tiempos de ejecución, en la Figura 11, se comparan dichos tiempos en los dos entornos multicore, visualizándose para la Imagen Lena, con unos valores de ruido gaussiano σ=10 e impulsivo ρ=0,1, con 16 iteraciones. En la Figura 12 se muestran los tiempos de ejecución para la Imagen Caravana en el Multicore 2, con unos valores de ruido gaussiano σ=20 e impulsivo ρ=0,2, con 17 iteraciones.

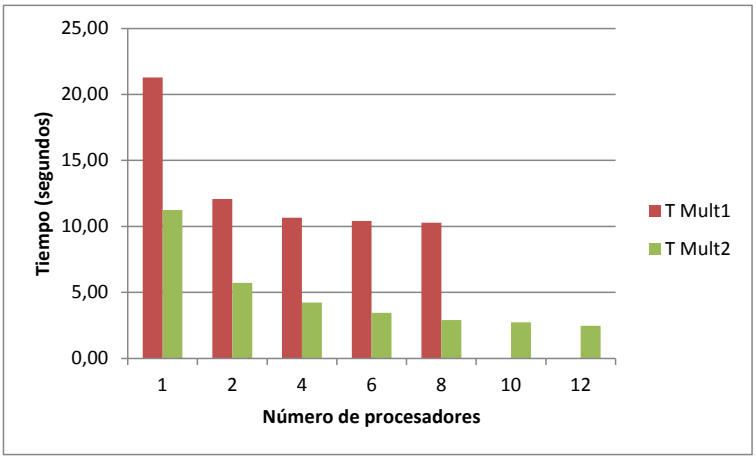


Figura 11. Comparativa de tiempos de ejecución en los dos entornos. Imagen Lena. (elaboración propia)

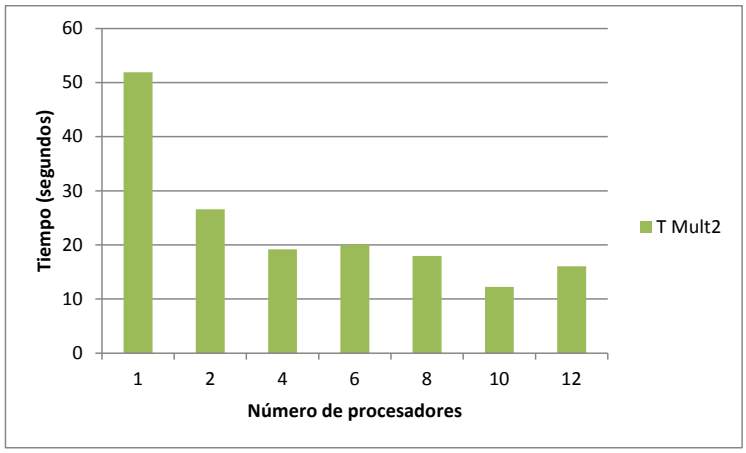


Figura 12. Tiempos de ejecución en el Multicore 2. Imagen Caravana. (elaboración propia)

Observando las dos imágenes se observa que la reducción de tiempo es significativa, más en el entorno Multicore 2 que en el entorno Multicore 1, por los motivos de potencia y antigüedad de los equipos. Las conclusiones son similares para todas las imágenes testeadas, observando que el número de procesadores óptimo a utilizar depende del tamaño de la imagen.

6. Conclusiones

En este trabajo se ha propuesto un algoritmo paralelo para la corrección de imágenes a color, corruptas con ruido gaussiano e impulsivo.

El algoritmo paralelo explicado se basa en el filtro de media de pesos ponderada basado en lógica fuzzy para imágenes en color con ruido mixto gaussiano-impulsivo introducido en [20].

Después de su ejecución en dos máquinas multicore distintas, se ha llegado a la conclusión de que el algoritmo propuesto presenta un *speedup* significativo en el procesamiento de imágenes grandes comparado con el algoritmo secuencial. En algunos casos, el *speedup* obtenido es casi lineal con respecto al número de procesadores utilizados.

Pese a ser un algoritmo que se ejecuta iterativamente, los tiempos de ejecución obtenidos demuestran que el algoritmo paralelo propuesto podría ser válido, con criterios de parada menos restrictivos, para aplicaciones que requieran procesamiento de imágenes en tiempo real.

7. Líneas de trabajo futuras

La intención de este trabajo es que se convierta, dentro de su contexto, en un principio para futuras investigaciones y desarrollos, ya sea de forma individual por el autor, o por medio de una colaboración entre miembros de la comunidad científica y/o académica.

Por esta razón, se esbozan las posibles líneas futuras de investigación sobre el algoritmo paralelo para la corrección de ruido mixto gaussiano-impulsivo en imágenes de color, basado en lógica fuzzy:

- Estudio de técnicas de paralelización para incrementar la calidad de las imágenes filtradas, obteniendo mejores resultados de PSNR.
- Reprogramar el algoritmo utilizando el estándar Message Passing Interface (MPI) [30], así como utilizando técnicas mixtas MPI-OpenMP.
- Aplicación del algoritmo para Computación Heterogénea, utilizando tanto elementos GPU de nVidia, como el Xeon Phi de Intel, para unir paralelización de datos y de tareas, respectivamente.
- A corto plazo, está prevista la aplicación del algoritmo paralelo para la restauración de imágenes y tomografías médicas, por lo que se haría una adaptación del mismo para imágenes en escala de grises y su uso posterior en imágenes 3D.

8. Referencias

1. **Plataniotis, K. N. y Venetsanoupoulos, A. N.** *Color Image Processing and Applications.* New York, USA : Springer-Verlag, 2000.

2. **Boucelet, C.** *Image Noise Models.* London : Academic Press, 2000. págs. 325-335.

3. **Tomasi, C. y Manduchi, R.** *Bilateral filtering for gray and color images.* Proceedings of the 6th International Conference on Computer Vision, ser. ICCV'98. Washington DC, USA : IEEE Computer Society, 1998. págs. 839-.

4. **Schulte, S. y otros.** *A new fuzzy-based wavelet shrinkage image denoising technique.* Proceedings of the 8th International Conference on Advanced Concepts For Intelligent Vision Systems, ser. ACIVS'06. Berlin, Heidelberg : Springer-Verlag, 2006. págs. 12-23.

5. **Li, X.** *On modeling interchannel dependency for color image denoising technique.* s.l. : Int. J. Imaging Syst. Technol., Oct. 2007. págs. 163-173. Vols. 17, nº 3.

6. **Dabov, K. y otros.** *Image denoising by sparse 3-D transform-domain collaborative filtering.* s.l. : Trans. Img. Proc., Aug. 2007. págs. 2080-2095. Vols. 16, nº 8.

7. —. *Color image denoising via sparse 3d collaborative filtering with grouping constraint in luminance-chrominance space.* Proceedings of the

International Conference on Image Proceeding, ICIP 2007, San Antonio, Texas, USA : IEEE, 2007. págs. 313-316.

8. **Smolka, B.** *Peer group switching filter for impulse noise reduction in color images.* s.l. : Pattern Recognit. Lett., 2010. págs. 484-495. Vols. 31, nº 6.

9. **Camarena, J. G. y otros.** *Fast detection and removal of impulsive noise using peer groups and fuzzy metrics.* s.l. : J. Visual Commun. Image Represent., 2008. págs. 20-29. Vols. 19, nº 1.

10. **Toprak, A. y Güler, I.** *Impulse noise reduction in medical images with the use of switch mode fuzzy adaptive median filter.* s.l. : Digital Signal Process., 2007. págs. 711-723. Vols. 17, nº 4.

11. **Schulte, S. y otros.** *A fuzzy impulse noise detection and reduction method.* s.l. : IEEE Trans. Image Process., 2006. págs. 1153-1162. Vols. 15, nº 5.

12. —. *A new fuzzy color correlated impulse noise reduction method.* s.l. : IEEE Trans. Image Process., 2007. págs. 2565-2575. Vols. 16, nº 10.

13. —. *Fuzzy two-step filter for impulse noise reduction from color images.* s.l. : IEEE Trans. Image Process., 2006. págs. 3567-3578. Vols. 15, nº 11.

14. —. *Fuzzy random impulse noise reduction method.* s.l. : Fuzzy Sets and Systems, 2007. págs. 270-283. Vols. 158, nº 3.

15. **Melange, T., Nachtegael, M. y Kerre, E. E.** *Fuzzy random impulse noise removal from color images sequences.* s.l. : IEEE Trans. Image Process, 2011. págs. 959-970. Vols. 20, nº 4.

16. **Camarena, J. G. y otros.** *Some improvements for image filtering using peer group techniques.* s.l. : Image Vision Comput., 2010. págs. 188-201. Vols. 28, nº 1.

17. **Morillas, S., Gregori, V. y Peris-Fajarnés, G.** *Isolating impulsive noise pixels in color images by peer group techniques.* s.l. : Comput. Vision Image Understanding, 2008. págs. 102-116. Vols. 110, nº 1.

18. **Camarena, J. G. y otros.** *Two-step fuzzy logic-based method for impulse noise detection in colour images.* s.l. : Pattern Recognit. Lett., 2010. págs. 1842-1849. Vols. 31, nº 13.

19. **Arnal, J., Sucar, L. B. y otros.** *Parallel filter for mixed Gaussian-impulse noise removal.* SPA 2013 : Conference Proceedings, IEEE Press, 2013. págs. 236-241.

20. **Camarena, J. G. y otros.** *A simple fuzzy method to remove mixed Gaussian-impulsive noise from color images.* s.l. : IEEE Transactions On Fuzzy Systems, 2013. págs. 971-978. Vols. 21, nº 5.

21. Página web del Congress on Numerical Methods in Engineering - CMN 2017. [En línea] [Citado el: 20 de Enero de 2017.] http://congress.cimne.com/cmn2017/eng/.

22. **UTI.** *Reference algorithm for computing peak signal to noise ratio of a processed video sequence with compensation for constant spatial shifts, constant temporal shift, and constant luminance gain and offset.* Recommendation UTI-T J.340 : International Telecommunication Union, 2010.

23. **MSwamy, Karthik.** PSNR Calculator. *Matlab Central.* [En línea] [Citado el: 20 de Enero de 2017.] http://es.mathworks.com/matlabcentral/fileexchange/27862-psnr-calculator.

24. **Chapman, B., Jost, G. y van der Pas, R.** *Using OpenMP. Portable Shared Memory Parallel Programming.* Cambridge, Massachusetts : The MIT Press, 2008.

25. Página web de OpenMP. [En línea] [Citado el: 20 de Enero de 2017.] http://www.openmp.org/.

26. **García Merayo, Félix.** *Lenguaje de Programación Fortran 90.* s.l. : Editorial Paraninfo, 1999.

27. **Passino, K. y Yurkovich, S.** *Fuzzy control.* Menlo Park, CA : Addison Wesley, 1998.

28. **Driankov, D., Hellendoorn, H. y Reinfrank, M.** *An introduction to fuzzy control, 2nd ed.* New York : Springer-Verlag, 1996.

29. **Cox, E.** *The fuzzy systems handbook, 2nd ed.* New York : Academic, 1999.

30. Página web de MPI-Message Passing Interface. [En línea] [Citado el: 20 de Enero de 2017.] http://mpi-forum.org/.

31. Imagen Busto. [En línea] [Citado el: 20 de Enero de 2017.] https://nihilnovum.files.wordpress.com/2010/01/antinous_louvre.jpg.

32. Imagen Lena. *University of Southern California.* [En línea] [Citado el: 20 de Enero de 2017.] http://sipi.usc.edu/database/database.php?volume=misc&image=12#top.

33. Imagen Caravana. [En línea] [Citado el: 20 de Enero de 2017.] https://www.flickr.com/photos/di_bisceglie/.

34. Imagen Zapatos. [En línea] [Citado el: 20 de Enero de 2017.] https://www.flickr.com/photos/innovchallenge/.

35. Imagen Wash-IR. *University of Southern California.* [En línea] [Citado el: 20 de Enero de 2017.] http://sipi.usc.edu/database/database.php?volume=aerials&image=38#top.

36. Imagen Typhoon Bopha. *Archivos NASA.* [En línea] [Citado el: 20 de Enero de 2017.] http://earthobservatory.nasa.gov/NaturalHazards/view.php?id=79872.